بسم الله الرحمن الرحيم

دوسية جافا 1

إعداد: أ.علي خالد العزة
كلية فلسطين الأهلية الجامعية
بريد الكتروني: ali@paluniv.edu.ps

أولاً: ما هي الجافا؟

الجافا هي عبارة عن لغة برمجة تستخدم مفاهيم **OOP(Object Oriented Programming) و تعتمد هذه المفاهيم على الأوبجكت و هو عبارة عن خواص(states) و سلوك (behavior) مثلا لو** أننا **نريد عمل اوبجكت لضوء نحتاج لمعرفة ماذا نحتاج من خواص الضوء و سلوكه.**
الضوء –> صفة->اللون
الضوء->سلوك->يضيء
تستخدم الجافا كومبايلر و الإنتربتر و كلها تنضم في ال Java Virtual Machine و يجب تنزيله على نظام التشغيل لتشغيل البرامج المكتوبة بلغة الجافا ولذلك تعمل الجافا على كل أنظمة التشغيل.

ثانياً:قواعد أساسية في برمجة الجافا:

يتكون برنامج الجافا بشكل أساسي من :
Methods و هي عبارة عن مجموعة من العمليات مكتوبة بلغة الجافا تنفذ عند استدعائها **و يقابلها في لغة C ال Functions و هناك أيضا ال Class و هو عبارة عن البرنامج المكتوب بلغة الجافا و يحتوي على مكونات البرنامج .**

مثال:

```
class HelloJava{
	public static void main(String[] args){
		System.out.println("مرحبا بكم في جافا");
	}
}
```

نستطيع كتابة أكثر من كلاس في البرنامج الواحد و لكن دائما هناك كلاس رئيسي يحتوي على الميثود الرئيسي (Main Method) و لا يحتوي أي من الكلاس على الميثود الرئيسي إلا الكلاس الرئيسي و للميثود الرئيسي اسم آخر حيث يسمى القائد (Driver Method) لأنه انطلاقا **منه ينفذ البرنامج مثل في لغة C .**

ملاحظة : يجب أن يكون أول حرف من اسم **الكلاس كبيراً و كل قسم منه يبدأ أيضا بحرف كبير لأن** الاسم **يكون متلاصق. مثل : HelloJava القسم الأول : Hello و القسم الثاني:Java بينما لا تبدأ أسماء الميثود بأحرف كبيرة و لكن تقون أقسامها الباقية تبدأ بأحرف كبيرة مثلا الميثود getMax القسم الأول get يبدأ بحرف صغير و لكن القسم الثاني Max يبدأ بحرف كبير و ذلك لفهم الإسم.**

System.out.println() **: يستخدم لطباعة الناتج حيث** إذا **استخدمنا:**
println **: يطبع ثم ينزل سطر.**
print **: يطبع بدون أن ينزل سطر.**

- أنواع المتغيرات المعرفة تلقائياً في الجافا(Built in Types) :

هناك عدة أنواع معرفة للمتغيرات في الجافا كبقية لغات البرمجة:

يكون تعريف المتغير كالتالي:

نوع المتغير		اسم المتغير
مثال	**char**	**x**

أنواع المتغيرات المعرفة في جافا:

Integers(الأعداد الحقيقية) :

كيفية كتابته عند البرمجة	الحجم (بيت)	مدى الأرقام أقل قيمة	أكبر قيمة
byte	8 bits	-2^{7}	$2^{7}-1$
short	16 bits	-2^{15}	$2^{15}-1$
int	32 bits	-2^{31}	$2^{31}-1$
long	64 bits	-2^{63}	$2^{63}-1$

Byte = 8 bits.

ملاحظة: عند تعريف long يجب كتابة l **(حرف أل)**بعد الرقم, مثال:

```
long x = 1234321l ;
```

Float(الأرقام العشرية):

كيفية الكتابة عند التعريف	الحجم (بيت)	المدى
float	32	7-6 أرقام بعد الفاصلة
double	64	15 رقم بعد الفاصلة

ملاحظة :عند تعريف فلوت يجب كتابة f بعد الرقم,مثال:

```
float x = 1.23f;
```

هناك كذلك للأرقام العشرية قيم خاصة مثل :

PositiveInfinite : $+\infty$
NegativeInfinite: $-\infty$
Nan (not a number): 1/0(غير معرف).

Boolean (تعبير جبرى له قيمتين صح او خطأ):

boolean Size : 1bit Values : true , false.

مثال:

```
boolean x = true;
```

Characters:

هي عبارة عن حرف واحد او رمز حجمه 2 بايت و يكون دائما تعريفه بوضع الحرف او الرمز بين ' ' أو بكتابة الرقم المقابل له بالأسكي كود و هي مجموعة الأرقام المعرفة للأحرف و الرموز.

char **Size:** 16bit

ملاحظة: نستطيع الكتابة بأي لغة نريدها في الجافا لأنها تستخدم Ascii code و الترميز الموحد Unicode .

مثال:

```
char x = 'A';     char x = 65;
char x='س';
char x='#';
```

65 يقابله الحرف A في الأسكي كود.

هناك أيضا الرمز '\n' و هذا يعني انزل سطر أو enter و يقابله في الأسكي كود رقم 13

Operators (المعاملات) :

+ , - , * , / , ++ , --

ملاحظة : ++ يعني تزيد للمتغير قيمة 1
-- تنقص من قيمة المتغير قيمة 1

مثال:

```
int x =4; x++;
```

النتيجة: x=5

Relations(علاقات):

تساوي و تستخدم عند السؤال : ==
لا تساوي و تستخدم أيضا عند السؤال : =!
أكبر من أو تساوي : =>
أقل من أو تساوي:=<
أكبر :>
أصغر : <

Boolean Operators:

و تستخدم عند السؤال
&& : and
|| : or
^^ : xor
! : not

Bitwise Operators:

و تستخدم في العمليات المنطقية
&: and
| : or
^ : xor
~ : not

<u>More Complicated :</u>

إزاحة الى اليمين : >>
إزاحة الى اليسار : <<
باقي القسمة : %

مثال:

```
int x;
x = 4%3;
```
الخروج : x= 1

<u>Automatic Conversion (التحويل التلقائى):</u>

تستطيع الجافا تحويل شتى انواع المتغيرات في نتائج العمليات المختلفة الى نوع واحد حسب قوانين خاصة حسب الأحقية.

مثال:

1.7 + 99 = 100.7
double + int = double
int→ double ثم قام بالعملية
double النتيجة

<u>حيث تكون الأولويات عند وجود انواع المتغيرات فى العملية بالترتيب التالى:</u>

1- إذا كان في العملية double تحول المتغيرات الأخرى الى double و النتيجة تكون من نوع double.
2- إذا كان في العملية float تحول المتغيرات الأخرى الى float و النتيجة تكون من نوع float .
3- إذا كان في العملية long تحول المتغيرات الأخرى الى long و النتيجة تكون من نوع long.
4- إذا لم تكن في العملية أي من الانواع السابقة تحول كل المتغيرات الى int و النتيجة تكون من نوع int.

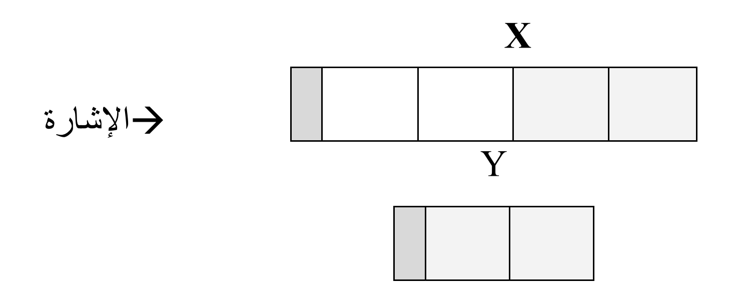
<u>Casting:</u>

هي عملية ارجاع قيمة نحن نختارها بعد القيام بالعملية.

مثال على casting:

```
int x = 2345;
short y = (short)x;
```

int حجمه 4 بايت و لكن short حجمه 2 بايت لذلك يأخذ لy 2 بايت من يمين x من القيمة الممثلة بالذاكرة و اذا كانت هناك اشارة يأخذ بالإضافة لذلك الإشارة



Strings(المتسلسلات):

هي عبارة عن مجموعة من الأحرف **(رموز + أحرف)** تستخدم كقطعة واحدة و يكون تعريفها بوضع مجموعة الرموز بين " " كالتالي :

```
String s = "Java 2";
أو
String s;
s = "جافا";
```

يجب مراعاة أن يكون الحرف الأول كبير في هذا النوع حيث يكون حرف ال S كبير.

ملاحظة: يمكن استخدام عملية اضافة أحرف أو جمل للمتغيرات من نوع String .

مثال:

```
String s = "Java ";
int x=2;
s+= "2";     أو     s = s+"2";   أو   s+= '2';  أو   s+=x;
                                                الخروج: s="java 2"
```

مثال كتابة برنامج يستخدم String:

```
public class Triangle{
	public static void main(String[] args){
		String s= "*";
		for (int i=0;i<=6;i++){
			System.out.println(s);
			s+="*";
		}
	}
}
```

Note: s+ = "*" تعادل s = s + "*";

خروج البرنامج:

```
*
**
***
****
*****
******
*******
```

بعض الميثود الخاصة ب String :

int length() : يرجع طول الِ String مثال:

```
String s = "Core Java 2";
int l = s.length();
```

الخروج : l=11

char charAt(int i): يرجع حرف في الموقع i من ال String مثال:

```
String s = "Core Java 2";
char c = s.charAt(3);
```

الخروج c='e' لأننا نبدأ عد الأحرف من الصفر في المتسلسلة.

0	1	2	3	4	5	6	7	8	9	10
C	o	r	e		J	a	v	a		2

الطول 11 حرف

مثال برنامج يعمل قلب لكلمة:

```
public class Revers{
	public static void main(String[] args){
		String s = "Java";
		int length = s.length();
		for (int i=length;i>=0;i--)
			System.out.print(s.charAt(i));
	}
}
```

الخروج : avaj

Flow Control:

التكرار:

كثير من الأحيان في الجافا وتكاد جميع لغات البرامج تستخدم التكرار لتكرار عملية واحدة أكثر من مرة و في الجافا كما في لغات البرمجة الأخرى أوامر لعمل تكرار للعمليات و هذه الأوامر:

1- for(**تنفيذ عملية** ; **السؤال** ; **تعريف متغيرات مساعدة في السؤال في التكرار**)
   العملية المراد تكرارها
2- while(السؤال)
   العملية المراد تكرارها
3- do{
   العمليات المراد تكرارها
   }while(السؤال);

السؤال: نكتبه لعمل حد للتكرار حيث يكون اما صح او خطأ(true or false) حيث يبقى تكرار العملية ساريا حتى يصبح السؤال نتيجته خطأ مثلا:

```
int x=0;
for(int i = 0 ;i<9;i++)
	x++;
```

السؤال:هل i أصغر من 9 اذا كان الجواب نعم يكمل التكرار حتى تصبح i أكبر أو تساوي 9 و الجزء الأخير في الأمر for [i++] ينفذ لزيادة قيمة i حتى يصبح السؤال خطأ ليتوقف التكرار و التكرار هنا ينفذ العملية x++ 9 مرات حيث تصبح قيمتها 9.

مثال:

```
while(x!=0)
i+=5;
```

و السؤال هنا عندما x لا تساوي 0 نفذ i+=5

كيفية تنفيذ أكثر من عملية واحدة في تكرار واحد:

نستطيع ذلك بحصر التكرار ب {} **(block)** حيث ينفذ التكرار كل العمليات التي داخل البلوك في نفس التكرار.

مثال:

```
int i=10,x=0;
whil(i!=0){
        x++;
        i--;
}
```

حيث ينفذ العمليتين التان داخل البلوك (++x و --i) في التكرار while حتى تصبح i تساوي صفراً.

<u>الفرق بين التكرارين while و do while :</u>

الفرق بسيط حيث ينفذ التكرار do while العمليات مرة واحدة على الأقل قبل أن يسأل السؤال.

مثال:

```
int x=0,i=1;
do{
        x+=5;
}while(i==0);
```

حيث ينفذ العملية x+=5 مرة على الأقل قبل السؤال.

<u>الشرط:</u>

نستطيع في الجافا استخدام شرط لتنفيذ عملية ما وهناك أمرين لعمل الشرط:

1-if(الشرط)
العملية

و نستطيع وضع أي شرط مثل في عمليات التكرار و يكون اما صحيح أو خاطئ**(Boolean)**.

مثال:

```
if ( i==5)
        x++;
```

كما نستطيع حصرها ب {} لتنفيذ أكثر من عملية.

نستطيع اتباعها ب else حيث اذا لم ينفذ العملية الأولى في الشرط ينفذ عملية اخرى.

مثال:

```
if(i==5)
        x++;
else
        x+=2;
```

اذا لم ينفذ ++x سينفذ x+=2 و نستطيع كذلك حصر else ب {} .

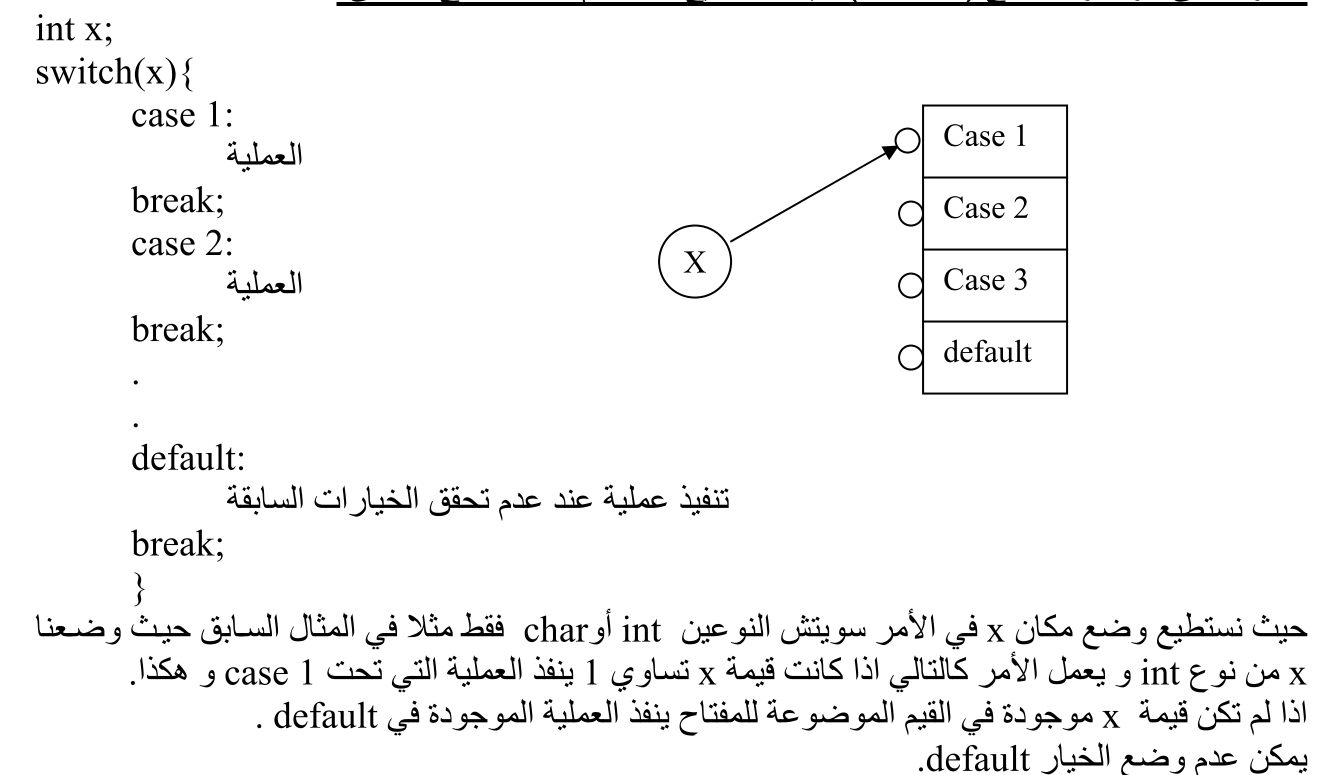
أو اتباعها ب else if(boolean) حيث ان لم ينفذ الشرط الأول يجرب الشرط الثاني.

مثال:

```
if(i>=4)
        x++;
else if(i>6)
        x*=6;
```

الأمر الثاني هو أمر المفتاح (switch) حيث نستطيع استخدام هذا المفتاح كالتالي:

```
int x;
switch(x){
        case 1:
                العملية
        break;
        case 2:
                العملية
        break;
        .
        .
        default:
                تنفيذ عملية عند عدم تحقق الخيارات السابقة
        break;
        }
```



حيث نستطيع وضع مكان x في الأمر سويتش النوعين int أو char فقط مثلا في المثال السابق حيث وضعنا x من نوع int و يعمل الأمر كالتالي اذا كانت قيمة x تساوي 1 ينفذ العملية التي تحت case 1 و هكذا.
اذا لم تكن قيمة x موجودة في القيم الموضوعة للمفتاح ينفذ العملية الموجودة في default .
يمكن عدم وضع الخيار default.
ينتهي كل case بالأمر break حيث ينهي كافة العمليات التي داخل ال case .
ملاحظة: يمكن استخدام الأمر break في التكرار و ذالك لإنهاء عملية التكرار.

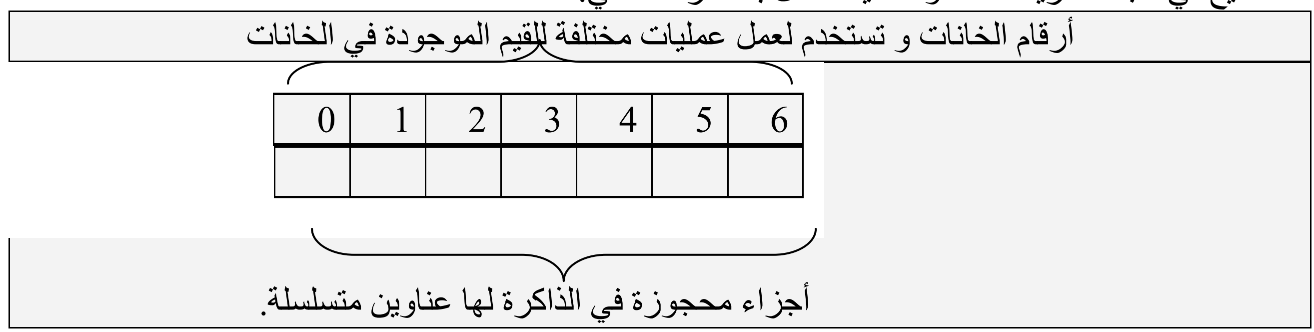
مثال:

```
while(x!=0){
        x++;
        if(x>0)
                break;
}
```

في هذا المثال يخرج من عملية التكرار اذا تنفذ شرط if .

Arrays(المصفوفات) :

نستطيع في الجافا تعريف مصفوفة حيث تمثل بالذاكرة كالتالي:



تحجز المصفوفات عناوين متسلسلة في الذاكرة و يكون حجم كل جزء بمقدار حجم النوع الذي عرفت به المصفوفة.
مثال:

```
int[] a = new int[10];
int[] a = {1,3,5,2};
```

هنا عرفنا المصفوفة بطريقتين الأولى عرفنا المصفوفة بأن بنينا مصفوفة جديدة من نوع int تحجز 10 خانات متسلسلة في الذاكرة و كل خانة لها حجم int (4byte) و لكن الخانات تحتوي على القيمة الافتراضية و هي صفر, اما في الثانية فعرفنا المصفوفة بأن وضعنا قيم لها حيث يحجز النظام خانات بعدد القيم الموضوعة و يكون حجمها من نوع int و كل خانة تحتوي على قيمة من القيم الموضوعة حسب التسلسل.
<u>ملاحظة :</u> الأمر new يعمل على بناء object حيث يحجز في الذاكرة مكان لأوبجكت جديد.

<u>كيفية استخدام المصفوفة و اخذ القيم منها:</u>
نستطيع معرفة القيمة المخزنة في أي خانة أو تخزين قيمة جديدة في أي خانة او عمل عمليات على هذه الخانة عن طريق العنوان (index) و هو رقم الخانة.
مثال:

```
int[] a = {1,3,5,6,2};
int x=a[0];
a[1]+=4;
```

في هذا المثال خزنا قيمة الخانة 0 في المتغير x حيث أصبحت قيمة x تساوي 1 .
و أضفنا للخانة 1 القيمة 4 فأصبحت a[1] تساوي 7.
نستطيع معرفة طول المصفوفة باستخدام الميثود int length كالتالي:

```
int x;
int[] e={2,5,3,7,1,9};
x=a.length;
```

حيث قيمة x تساوي بعد العملية الأخيرة 6 و هو عدد الخانات المحجوزة للمصفوفة.

مثال برنامج يوجد أصغر رقم موجود في المصفوفة:

```
Class Minimum{
        public static void main(String[] args){
                int[] a={3,8,4,9,1};
                int min = a[0];
                for(int i=0;i<a.length;i++){
                        if(a[i]<min)
                                min=a[i];
                }
                System.out.println("the min number in the Array = "+min);
        }
}
```

الخروج: min = 1.

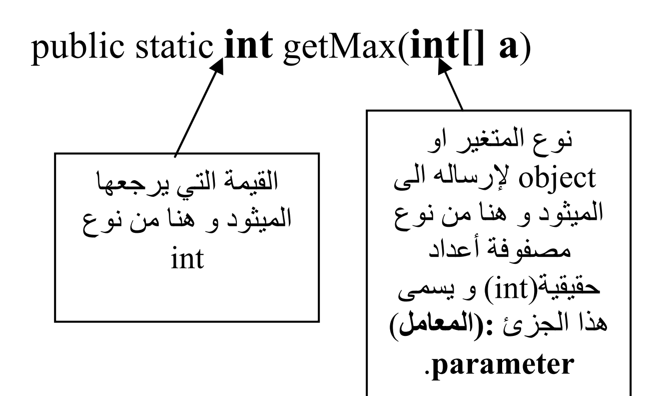
إستخدام ال Method في برمجة جافا:

نستطيع كتابة أكثر من ميثود غير الميثود الرئيسي في برنامج واحد و يتم استدعاء كل ميثود عند الحاجة من الميثود الرئيسي.

مثال حساب أعلى قيمة موجودة في المصفوفة:

```
class Maximum{
        public static int getMax(int[] a){ميثود لحساب اعلى قيمة
                int max = a[0];
                for(int i=0;i<a.length;i++){
                        if(a[i]>min)
                                max=a[i];
                }
                System.out.println("the min number in the Array = "+min);
        }

        public static void main(String[] args){هنا الميثود الرئيسي
                int[] b={2,5,4,9,1,33};
                int max=getMax(b);هنا نادينا الميثود
                System.out.println("the max number in the Array = "+max);
        }
}
```



ملاحظة: لايهم ترتيب الميثود في الجافا حيث يمكن أن يكون الميثود الرئيسي في الآخر.
يجب ان تكون الميثود التي هي خارج الأوبجكت من نوع static لنناديها كما في المثال السابق.

2D Arrays(المصفوفات ذات الإتجاهين) :

نستطيع تعريف مصفوفة ذات اتجاهين في الجافا و تمثل في الذاكرة بنفس تمثيل المصفوفة ذات الإتجاه الواحد.
مثال تعريف المصفوفة ذات الإتجاهين:

```
int[][] a = new int[2][4];
أوبوضع لها قيم
int[][] a = {{1,2,4,7},{-1,5,8,9}};
```

للوصول لقيمة خانة في المصفوفة نضع رقم الصف و رقم العامود المراد.

مثال:

```
int[][] a= {{1,2,4,7},{-1,5,8,9}};
int x = a[0][3];
```

هنا أصبحت قيمة x تساوي 7.

1	2	4	7
-1	5	8	9

Comments and Documentation(الملاحظات و التعليقات):

هناك نوعين من التعليقات في الجافا حيث تكتب هذه التعليقات أو الملاحظات لكي يستطيع المبرمجون الآخرون فهم ما كتبناه من ميثود أو لكي يستطيع المستخدم فهم البرنامج ولا تؤثر أبدا في عمل البرنامج و هذان النوعان هما:

Comments :

و تكتب بعد الإشارة // أو تحصر بين /*......**التعليق**.....*/ و تكتب ليستطيع المبرمج قراءة البرنامج الذي كتبناه و فهمه.

Documentation:

و تكتب بين /*......**المساعدة**........**/ والفائدة منها لجعل المستخدم يفهم ماذا يعمل البرنامج.

ملاحظة: هناك أدوات في جافا تستطيع قراءة التعليقات و عمل منها صفحة انترنت و تسمى **javadoc** .

مثال:

```
/**class to print star
   @auther : Ali azzeh
   @company : al quds university
   @version 1.0 */
class star{
        public static void main(String[] args){
                System.out.print("*");              //print star
        }
}
```

في الجافا يجب كتابة التعليقات لأهميتها حيث في قوانين البرمجة بالجافا يجب مراعاة الترتيب و التنظيم للبرنامج و عدم استخدام side effect و يعني عدم كتابة أكثر من عملية في سطر واحد.

مثال:

```
i=23+22;b=13%67;                side effect
```

و الحكمة من ذلك معرفة موقع الخطأ بالضبط عند حصوله عند تنفيذ البرنامج لذلك يجب وضع كل عملية في سطر وحدها.

كما يجب مراعاة الترتيب في حصر الميثود أو الأوامر أو الكلاس لتسهيل فهم البرنامج.

مثال:

```
class star
{
|        public static void main(String[] args)
|       {
|        |        System.out.print("*");
|        }
}
```

ملاحظة: تكتب ال Documentation دائما قبل الميثود و ال field و ال Constructor.

ثالثاً:كيفية البرمجة بتقنية OOP :

الفكرة: ال object هو مجموعة من المعلومات.

Fields:

و هي عبارة عن المعلومات التي نحتاجها لعمل البرنـامج و هـي من صـفات الأوبجكت مثلا لـو أننـا أردنـا بنـاء أوبجكت للطالب نحتاج معلومات عن الطالب مثل اسمه و رقمه الجامعي و علامته.

مثال:

```
class Student{
        /**Stringاسم الطالب و هو من نوع*/
        public String name;
        /**doubleالرقم الجامعي و هو من نوع*/
        public double id;
         /**int العلامة و هي من نوع*/
        public int mark;
}
```

Constructor:

هو عبارة عن باني الأوبجكت عندما نحتاج بناء أوبجكت يجب علينا استخدامه حيث يعمل لنا أوبجكت جديد لـه صفات الأوبجكت الذي بني منه.
يكون اسم ال Constructor نفس اسم الكلاس المعمول فيه.

مثال:

```
class Student{
        /**Stringاسم الطالب و هو من نوع*/
        public String name;
        /**doubleالرقم الجامعي و هو من نوع*/
```

```
        public double id;
         /*int العلامة و هي من نوع*/
        public int mark;
        /**Constructor of Student Object*/
        public Student(){
        }
}

class StudentTest{
        public static void main(String[] args){
                Student ali = new Student();
                ali.name ="ali";
                ali.id = 20011164;
                ali.mark = 80;
                System.out.print(ali.name+ali.id+"["+ali.mark+"]");
        }
}
```

يمكن أن يكون ال Constructor مع باراميتر ويمكن كذالك أن يكون هناك أكثر من Constructor لأوبجكت واحد و يكون الأختلاف بعدد الباراميتر و تسمى هذه الحالة **Overloading**.

يسمى الConstructor الذي ليس له باراميتر الباني الإفتراضي(Default Constructor) حيث يعمل بناء للأوبجكت دون اعطاء قيم للFields و يكون دائما مكتوب.

مثال:

```
class Student{
        /**Stringاسم الطالب و هو من نوع**/
        public String name;
        /**doubleالرقم الجامعي و هو من نوع**/
        public double id;
         /*int العلامة و هي من نوع*/
        public int mark;
        /**Default Constructor of Student Object*/
        public Student(){
        }
        /**Constructor of Student Object with parameter */
        public Student(String aName,double aId,int aMark){
                name = aName;
                id = aId;
                mark = aMark;
        }
}
class StudentTest{
        public static void main(String[] args){
                Student ali = new Student("ali",20011164,80);
```

```
                System.out.print(ali.name+ali.id+"["+ali.mark+"]");
        }
}
```

في الجافا التركيز دائما على الأوبجكت حيث هو الذي يحتوي على المعلومات و الميثود المطلوبة لتعطينا نتائج هذه المعلومات.
الميثود الرئيسي في الجافا هو عبارة عن أوامر لتنفيذ الميثود التي داخل الأوبجكت فقط و لا تستخدم مفاهيم الفنكشن في الجافا مثل في لغة c .

Public / Private:

Public:
عند وضع كلمة public قبل الميثود أو الفيلد أو باني الأوبجكت نستطيع استخدامهم في أي كلاس آخر و في هذه الحالة يجب كتابة ال documentation لهذه الأشياء و ذلك يجعل امر تحديثها امرا صعبا لأن ممكن أي تحديث لها أن يغير مفهومها و بذلك يصبح التعليق الذي كتب عنها خاطئ فلا يعرف المستخدم ماذا تعمل هذه الأشياء بالضبط.

Private:
اذا كانت الأشياء السابقة(ميثود,فيلد,....) private لا نستطيع استخدامها الا من الكلاس التي عرفت فيه و لا نحتاج لكتابة التعليق و بذلك يكون أمر تحديثها سهل و لذالك نستخدم دائما في الأوبجكت فيلد معرفة private و ميثود معرفة public .

Accessor and Mutator:

في الأوبجكت دائما هناك لكل فيلد معرف ميثودين و هما :

Accessor:
و يستخدم للوصول للقيمة المخزنة داخل هذا الفيلد و يبدأ دائما اسمه بكلمة get ويرجع قيمة الفيلد و هو بدون باراميتر.

Mutator:

و يستخدم لتغيير قيمة فيلد معين و يبدأ دائما اسمه بكلمة set وله باراميتر و لكن لا يرجع قيمة .

مثال:

```
class Student{
        private String name;
        private double id;
        private int mark;
        /**Default Constructor of Student Object*/
        public Student(){
        }
        /**Constructor of Student Object with parameter */
        public Student(String aName,double aId,int aMark){
```

```
		name = aName;
		id = aId;
		mark = aMark;
	}
	/**accessor to name*/
	public String getName(){
		return name;
	}
	/**mutator to name*/
	public void setName(String aName){
		name = aName;
	}
		/**accessor to Id*/
	public double getId(){
		return id
	}
	/**mutator to Id*/
	public void setId(double aId){
		id = aId;
	}
		/**accessor to Mark*/
	public int getMark(){
		return mark;
	}
	/**mutator to Mark */
	public void setMark(int aMark){
		mark = aMark;
	}
}

class StudentTest{
	public static void main(String[] args){
		Student ali = new Student();
		ali.setName("ali");
		ali.setId(20011164);
		ali.setMark(80);
		System.out.print(ali.getName()+ali.getId()+ali.getMark());
	}
}
```

الخروج : ali 20011164 80 .

ملاحظة: عند وضع كلمة void قبل الميثود هذا يعني أنه لا يرجع قيمة.

toString Method:

هو عبارة عن ميثود يرجع لنا معلومات عن الأوبجكت و القيمة التي يرجعها من نوع String.

عند بناء الأوبجكت يكون موجود هذا الميثود افتراضيا و لكن عند مناداته يرجع لنا عنوان في الذاكرة و لذلك نحتاج عادة لإعادة كتابته.

```
class Student{
	private String name;
	private double id;
	private int mark;
	/**Default Constructor of Student Object*/
	public Student(){
	}
	/**Constructor of Student Object with parameter */
	public Student(String aName,double aId,int aMark){
		name = aName;
		id = aId;
		mark = aMark;
	}
	/**accessor*/
	public String getName(){
		return name;
	}
	/**mutator*/
	public void setName(String aName){
		name = aName;
	}
		/**accessor*/
	public double getId(){
		return id
	}
	/**mutator*/
	public void setId(double aId){
		id = aId;
	}
		/**accessor*/
	public int getMark(){
		return mark;
	}
	/**mutator*/
	public void setMark(int aMark){
		mark = aMark;
	}

	/**toString Method*/
```

```
        public String toString(){
                return "Name"+name+","+"ID:"+id+","+"MARK:"+mark;
        }
}
class StudentTest{
        public static void main(String[] args){
                Student ali = new Student("ali",20011164,80);
                System.out.print(ali.toString());
        }
}
```

الخروج: Name: ali , ID: 20011164 , MARK: 80

اذا كتبنا في أمر الطباعة بدل ali.toString فقط ali أي اسم الأوبجكت يستخدم الميثود toString تلقائياً .

مثال:

```
System.out.print(ali);
هي نفسها:
System.out.print(ali.toString());
```

الخروج: Name: ali , ID: 20011164 , MARK: 80

ملاحظة:اذا كان الفيلد من نوع static فتكون قيمته ثابتة لكل الأوبجكت المعرفة من نفس الكلاس و لكن يمكن تغيير قيمته عند الحاجة.

مثال:

```
class Student{
        private String name;
        private double id;
        private int mark;
        private static int nextId=5;
        .
        .
        .
}
class StudentTest{
        public static void main(String[] args){
                Student ali = new Student("ali",20011164,80);
                Student hamzeh = new Student("hamzeh",20111159,80);
                System.out.print(ali.getNextId()+" , "hamzeh.getNextId());
        }
}
```

الخروج: 5 , 5.

Constant(الثابت) :

نستطيع تعريف فيلد ثابت القيمة لا يمكن تغيير قيمته في البرنامج و انما يستخدم كثابت و ذلك باستخدام الكلمة المعرفة final.

مثال:

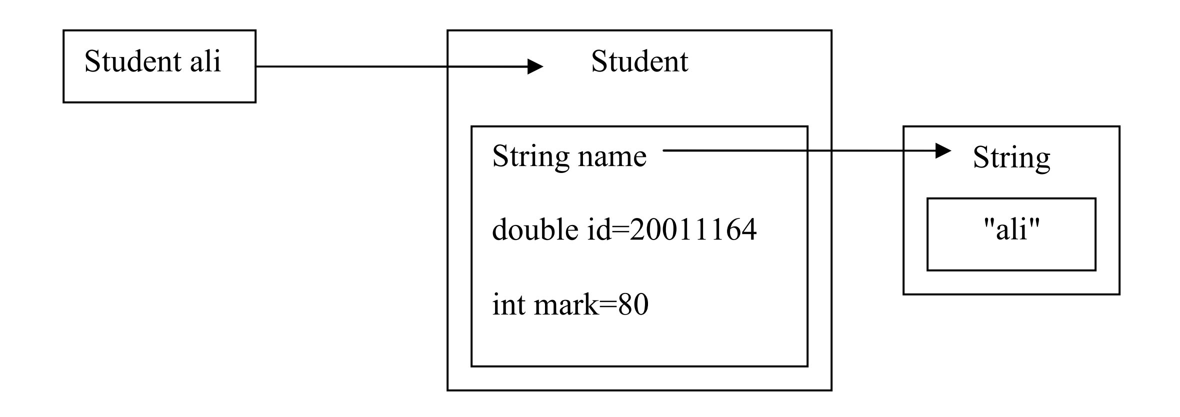
```
class StudentTest{
```

```
        public static final String a="NAME :  ";
        public static void main(String[] args){
                Student ali = new Student("ali",20011164,80);
                Student hamzeh = new Student("hamzeh",20111159,80);
                System.out.print(a+ali.getName());
        }
}
```

الخروج: NAME : ali .

تمثيل الأوبجكت في الذاكرة:

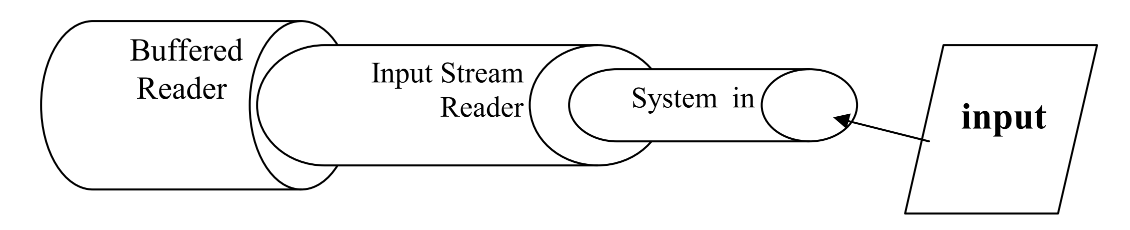
يمكن رسم رسم تمثيلي للأوبجكت في الذاكرة مثلا لرسم الأوبجكت ali في المثال السابق:



ملاحظة : ال String هو عبارة عن أوبجكت لذلك رسمنا سهم من اللإسم الى الأوبجكت String .

Input and Output:

في الجافا أدوات الإدخال و الإخراج تسمى مجرى(Stream) حيث يتم استقبال المعلومات المدخلة عن طريق مجرى كالتالي:



القرائة من لوحة المفاتيح:

للقرائةمن لوحة المفاتيح نحتاج 3 أنابيب(3 Stream) كما في الصورة التي في الأعلى و هم:

System in:

و هو الأنبوب الأول و يعمل على قرائة بايت واحد في كل مرة.

Input Strem Reader:

و يعمل على تحويل كل 2 بايت الى حرف أو رمز.

Buffered Reader:

و يعمل على تجميع هذه الحروف أو الرموز في ذاكرة مؤقتة لعمل منها سلسلة(String).

مثال:

```
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
```

هنا القارئ in هو من نوع الكلاس BufferedReader و الكلاس BufferedReader له المعامل(parametr) InputStremReader و الكلاس InputStreamReader له المعامل System.in و هو كلاس للقرائة بايت من لوحة المفاتيح.
و هي مع بعضها كما في المثال تمثل مجرى لقرائة حرف أو سلسلة من الأحرف من لوحة المفاتيح.

ملاحظة : in هو مجرد اسم يمكن تغييره.

Try And Catch:

عند عمليات القرائة و الكتابة في الجافا تحصر أوامر القرائة و الكتابة بين الكلمتين try و catch و تعني جرب القراءة أو الكتابة و اذا لم تنجح فأمسك الخطأ الذي تسبب في عدم النجاح في العملية , و الأخطاء تسمى "Exception" و هناك أنواع عديدة مثل : IOException و تعني خطأ في اجهزة الدخول و الخروج أو مثلا NumberFormatException و هي أخطاء التحويل من String الى رقم(int,long,…..).
مثال:

```
try{
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    String s = in.readLine();  //يقرأ سطر من لوحة المفاتيح
    }catch(Exception e){
                                      .......ماذا نريد أن يفعل إذا كان هناك خطأ في العملية......
                                    }
```

e هي خطأ عام أي لم نحدد أي نوع من الأخطاء يجب أن يمسك البرنامج.
e هو اسم يمكن تغييره.
نستطيع طباعة الخطأ على الشاشة هكذا:

```
try{
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    String s = in.readLine();
    }catch(Exception e){
                                      System.out.print(e); //يطبع لنا الخطأ بالضبط
                                    }
```

ملاحظة1: يتم تعريف القارئ مرة واحدة في الكلاس , مثلا في المثال السابق القارئ in يتم تعريفه مرة واحدة في الكلاس.

ملاحظة2: استخدام Tryو catch ليس محصوراً بعمليات الإدخال و الإخراج و إنما يستخدم لحصر أي عملية يمكن أن تحدث خطأ.

القراءة من ملف:

نستطيع القرائة في الجافا من ملف نصي (text file) عن طريق الكلاس FileReaderحيث يكون تعريف القارئ كالتالي:

```
BufferedReader r = new BufferedReader(new FileReader("c:\myFile.txt"));
```

المعامل (parameter) للكلاس FileReader هو عبارة عن سلسلة و تعبر عن المسار الذي موجود فيه الملف مثلا:

```
"c:\myDocument\doc\ملفي.txt"
```

مثال برنامج يعد الأسطر في ملف:

```
class CountFileLine{
        public static void main(String[] args){
          try{
           BufferedReader r = new BufferedReader(new FileReader("c:\myFile.txt"));
           String s;
           int count=0;                          //int متغيرلحساب الأسطر و هو من نوع
           while((s = r.readLine())!=null)     //يقرأ سطر سطر حتى نهاية الملف
              count++;
           r.close();                          //لإغلاق الملف
           System.out.print("The Number Of line in this file is : "+count);
              }catch(Exception ee){
                                          System.out.print("error when reading file : \n"+ee);
                                         }
        }
}
```

<u>**ملاحظة:**</u> null هي كلمة معرفة في الجافا و تعني **لا شيء**.

<u>**الكتابة في ملف:**</u> هناك العديد من الأنواع من الكلاس التي تستخدم للكتابة و منها الكلاس PrintWriter و هو يسمح بالتعديل على الملف كذلك و هو يستخدم كذلك الكلاس FileWriter كمعامل و المعامل الآخر هو من نوع Boolean حيث اذا وضعنا القيمة true يعمل اضافة على الملف المطلوب اما اذا كانت false فلا يقوم بالإضافة و انما يمسح ما في الملف السابق و يكتب غيره.

مثال برنامج يقرأ من لوحة المفاتيح 10 أرقام و يخزنها في ملف:

```
public class PrintWriterTest {
   public static void print(int[] a){
      try{
           PrintWriter out=new PrintWriter(new FileWriter("c:\ out.txt",false));
           for (int i=0;i<a.length;i++)
                out.println(a[i]);
                out.close();
         }catch(IOException e){System.out.println(e);}
```

```
    }

    public static void main(String[] args){
        int[] a=new int[10];
        String l;
         BufferedReader b=new BufferedReader(new InputStreamReader(System.in));
         System.out.println("ENTER TEN NUMBER: ");
         try{
             for (int i=0;i<a.length;i++){
                  l=b.readLine();
                  a[i]=Integer.parseInt(l); //يحول سلسلة الى عدد حقيقي
              }
           }catch(Exception d){System.out.println(d);}
         print(a);
    }
}
```

int Integer.parseInt(String s) :

يعمل على تحويل ال String s الى رقم اذا كانت السلسلة عبارة عن رقم و الا يكون هناك Exception أي خطأ مثلا:

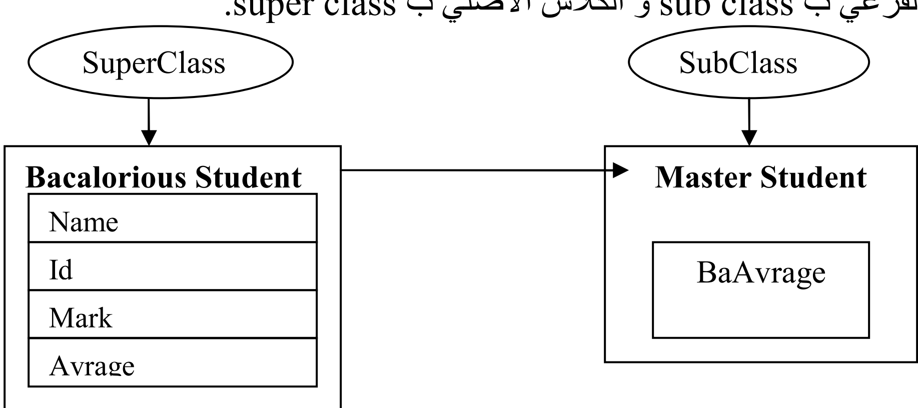
String s = "abc";
String d="123";
int l = Integer.parseInt(s); //error NumberFormatException
int a = Integer.parseInt(d); // true and a = 123

كما يوجد هناك ميثود لتحويل من السلسلة الى قيم اخرى مثل:

float Float.parseFloat(String s) //يحول الى رقم عشري
long Long.parseLong(String s)//يحول الى رقم من نوع لونج

Inheritance (التوارث):

و هو عمل كلاس او اوبجكت فرعي من كلاس آخر مثلا هناك نوعين من الطلاب طلاب ماجستير و طلاب بكالوريوس و الإثنان في النهاية هم طلاب و طالب الماجستير مثلا يختلف عن طالب البكالوريوس بإضافة بعض المواصفات لذلك نستطيع عمل أوبجكت فرعي لطلاب الماجسنير من أوبجكت طلاب الباكالوريوس حيث يسمى الكلاس الفرعي ب sub class و الكلاس الأصلي ب super class.



في هذه الصورة كل الطلاب لهم اسم و رقم جامعي و علامة و معدل تراكمي و لكن لطلاب الماجستير لهم معدل اضافي لذالك و ضعنا لهم اوبجكت فرعي من الأوبجكت الرئيسي و هو الطلاب.

تستخدم كلمة extends لعمل كلاس فرعي كالتالي:

```
class MasterStudent extends Student{

}
```

(Student ← اسم الكلاس الرئيسي)

مثال:

الكلاس الرئيسي(super class):

```
class Student{
	private String name;
	private double id;
	private int mark;
	/**Default Constructor of Student Object*/
	public Student(){
	}
	/**Constructor of Student Object with parameter */
	public Student(String aName,double aId,int aMark){
		name = aName;
		id = aId;
		mark = aMark;
	}
	/**accessor to name*/
	public String getName(){
		return name;
	}
	/**mutator to name*/
	public void setName(String aName){
		name = aName;
	}
		/**accessor to Id*/
	public double getId(){
		return id
	}
	/**mutator to Id*/
	public void setId(double aId){
		id = aId;
	}
		/**accessor to Mark*/
```

```
	public int getMark(){
		return mark;
	}
	/**mutator to Mark */
	public void setMark(int aMark){
		mark = aMark;
	}
}
```

الكلاس الفرعي(sub class):

```
class MasterStudent extends Student{
	double BaAvrage;

	public MasterStudent(){
	}
	public MasterStudent(String aName,double aId,int aMark,double aBaAvr){
		super(aName,aId,aMark);  //هنا نستخدم الباني في الكلاس الرئيسي الذي له نفس المعاملات
		BaAvrage = aBAvr;
	}

	public void setBaAvrage(double average){
		BaAvrage = average;
	}
	public double getBaAvrage(){
		return BaAvrage;
	}
	public String toString(){
	    return "Master Student :\n"+super.toString+"BaAvrage= : "+getBaAvrage;
	}
}

class StudentTest{
	public static void main(String[] args){
		Student ali = new Student("ali",20011164,80);
		MasterStudent amjad = new MasterStudent("amjad",9711123,84,78);
		System.out.print(ali.toString()+"\n"+amjad.toString());
	}
}
```

this and super:

الدالة super تدل على الكلاس الرئيسي أما الدالة this فتدل على الكلاس التي هي موجودة فيه.
و نستطيع استخدام مثلا الدالة super بطريقتين :
1- في الباني(constructor) حيث يسمح استخدامها في السطر الأول فقط و تكتب كالتالي:
super(المعاملات)
حيث يذهب الى الباني في الكلاس الرئيسي و يستخدم الباني الذي له نفس المعاملات كما في المثال السابق.
2- في الميثود حيث نستطيع استخدام الميثود من الكلاس الرئيسي في الكلاس الفرعي باستخدام هذه الدالة كالتالي:
super.اسم الميثود
كما في المثال السابق في الميثود toString في الكلاس MasterStudent .
و نستطيع استخدام الدالة this كذلك بنفس الطريقة لكن الدالة this تستخدم داخل الكلاس فقط حيث إذا وضعناها في الباني كالتالي:
this(المعاملات);
يذهب الى الباني الذي له نفس المعاملات في الكلاس نفسه.
كما نستطيع كتابتها في الكونستراكتور و تدل على الأوبجكت الذي أرسل مثلا:

```
public Student(String name){
        this.name = name;
}
```

حيث لو اننا بنينا اوبجكت في المين ميثود كالتالي :
Student ali = new Student("ali");
تدل this في الباني على الفيلد name للأوبجكت ali و بذلك يستطيع التفريق بين الفيلد و الباراميتر.

بعض مميزات الجافا:

Overriding(الهيمنة):
نستطيع في الجافا كتابة نفس الميثود و له نفس المعاملات في الكلاس الفرعي الموجود في الكلاس الرئيسي.

Polymorphism(التعددية):

نستطيع في الجافا عمل مرجعية للأوبجكت مثلا في المثال السابق نستطيع عمل مساواة بين أوبجكت من نوع الكلاس الرئيسي بأوبجكت من نوع الكلاس الفرعي كالتالي:

```
MasterStudent y = new MasterStudent();
Student x = y;
```

هنا لا نستطيع استخدام الميثود الموجودة داخل الكلاس الفرعي للأوبجكت x لأنه من نوع الكلاس الرئيسي.
نستفيد من هذه الخاصية بعمل مصفوفة من الطلاب.
مثال:

```
Student[] std = new Student[2];
std[0] = new Student();
std[1] = new MasterStudent();
```

Dynamic Binding(الربط الديناميكي):
يستخدم الجافا هذه الخاصية حيث إذا عملنا مساواة مثل في المثال السابق يكون المرجعية في استخدام الميثود للأوبجكت و ليس للمتغير.
حيث في المثال السابق لو وضعنا أمر طباعة كالتالي:

System.out.print(x);

يستخدم الميثود toString للأوبجكت y و ليس للمتغير x.

Constant Class(الكلاس الثابت):

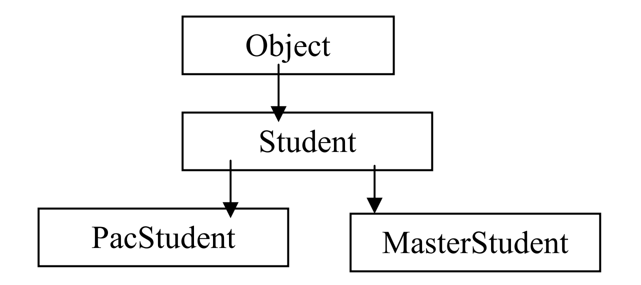
و هذا يعني أننا لا نستطيع عمل كلاس فرعي لهذا الكلاس يعني لا نستطيع عمل له overriding .
و لجعل الكلاس ثابت نكتب قبله كلمة final.
مثال:

final class Student{…}

حيث لا نستطيع عمل كلاس فرعي مثل الكلاس MasterStudent من الكلاس Student لأنه ثابت .

Hirachies(الهرم):

في الجافا كل الكلاسات هي عبارة عن كلاسات فرعية للكلاس أوبجكت.



Abstract Classes (الكلاسات المجردة):

عندما نعرف كلاس abstract هذا يعني أننا لا نستطيع استخدامه كأوبجكت ولا نستطيع استخدام الميثود التي يداخله إلا من خلال الكلاسات الفرعية مثلا لو نريد عمل برنامج لاستئجار الكتب من مكتبة الجامعة و هناك نوعين من المستخدمين الأساتذة و الطلاب لذلك نعمل اوبجكت abstract للمستخدمين عامة و اوبجكت فرعية لكل من الأساتذة و الطلاب.
مثال:

```
abstract class User{
        private String name;
        private String bookName;
        private Date borrowDate;
        public User(String name){
                this.name = name;
        }
        public String getName(){
                return name;
        }
        public void borrowBook(String bName){
                bookName = bName;
                borrowDate = new Date();
        }
        public void returnBook(){
                bookName = null;
        }
```

```
	public String getBook(){
		return bookName;
	}
	public Date getBorrowDate(){
		return borrowDate;
	}
	public abstract boolean isLate(); //كتب هذا الميثود ابستراكت و هذا يعني انه يجب كتابته في الكلاسات الفرعية

}
```

نبدأ بكتابة الكلاس الفرعي:

```
class Student extends User{
	private int Id;
	public Student(String aName,int Id){
		super(aName);
		this.Id = Id;
	}
	public boolean isLate(){
		if (new Date().compareTo(borrowBook)>1)
			return true;
		return false;
	}
	public String getName(){
		return super.getName();
	}
	public int getId(){
		return Id;
	}
}
class Professor extends User{
	public Professor(String name){
		super(name);
	}
	public boolean isLate(){
		if (new Date().compareTo(borrowBook)>1)
			return true;
		return false;
	}
	public String getName(){
		return "Dr."+super.getName();
	}
}
class LibraryTest{
	public static void main(String[] args){
```

```
            User[] users = new User[3];
            users[0] = new Student("ali",20011164);
            users[0].borrowBook("Java 2 V2");
            users[1] = new Professor("وائل");
            users[1].borrowBook("JSP and Servlet");
            users[2] = new Student("سميح",9912312);
            for(int i = 0;i<users.length;i++){
                System.out.print("\n "+users[i].getName());
                If ( !(users[i].getBorrowBook().equals(null))){
                  System.out.print("\n"+users[i].getBorrowBook()+"has borrowd
                  If(users[i].isLate())
                       System.out.print("\n Hi Is Late!!!");
                }
            }
      }
}
```

بعض الأمور الغير واضحة في البرنامج في المثال السابق:

Date: هو عبارة عن كلاس يقرأ تاريخ و ساعة الجهاز

مثال:

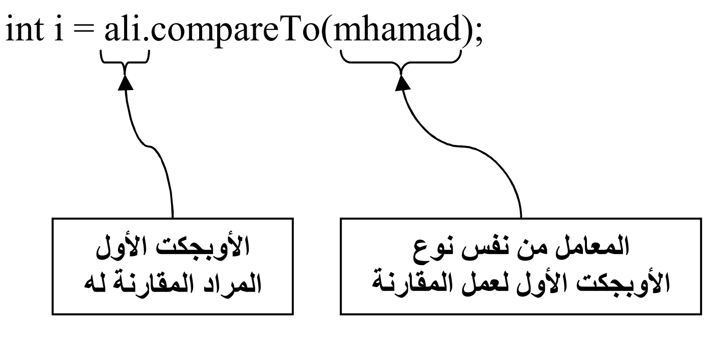
Date a = new Date(); //عند بناء اوبجكت جديد من هذا النوع يقرأ تاريخ و ساعة الجهاز

compareTo :

هو عبارة عن ميثود يعمل مقارنة بين اوبجكتين من نفس النوع و يرجع قيمة من نوع int حيث تعبر عن كبر او صغر الأوبجكت عن الآخر و تكون القيمة التي يرجعها 0 اذا كان الأوبجكتين متساويين.

مثال:

int i = ali.compareTo(mhamad);



equals:

هو عبارة عن ميثود يعمل مقارنة بين اوبجكتين من نفس النوع و يرجع قيمة من نوع boolean حيث اذا كانا متساويين يرجع true و الا يرجع false .

مثال:

boolean b = ali.equals(mhamad);

بعض الأوامر المفيدة:

System.exit(1): يعمل خروج من البرنامج

من أوامر الString:

String toLowerCase(): يعمل على تحويل احرف الكلمة الى احرف صغيرة

مثال:

```
String s = "GhOsT";
String h = s.toLowerCase();
```

الخروج : "h = "ghost

String toUpperCase() : يعمل على تحويل الأحرف الى أحرف كبيرة

int compareToIgnoreCase(String s) : يقارن الكلمة مع الكلمة أخرى بغض النظر عن نوع الحرف(كبير أو صغير)يرجع 0 اذا كانتا الكلمتين متساويتين

مثال:

```
String s = "GHOST";
String h = "ghost"
Int l = s. compareToIgnoreCase(h);
```

الخروج : l = 0

String substring(int beginIndex,int endIbdex):

يعمل على أخذ جزئ من السلسلة و نقو بتحديد قيم beginIndex موقع بداية قطع الكلمة و endIbdex و تحدد نهاية القطع من الكلمة.

مثال:

```
String s = "muhamad";
String sub = s. substring(2,6);
```

الخروج : "sub = "hamad